Natural SQL Statements

This section covers points you have to consider when using Natural SQL statements with DB2. These DB2-specific points partly consist in syntax enhancements which belong to the Extended Set of Natural SQL syntax. The Extended Set is provided in addition to the Common Set to support database-specific features.

This section covers the following topics:

- Common Syntactical Items
- CALLDBPROC
- COMMIT
- DELETE
- INSERT
- PROCESS SQL
- READ RESULT SET
- ROLLBACK
- SELECT
- UPDATE

Common Syntactical Items

The following syntactical items are either DB2-specific and do not conform to the standard SQL syntax definitions (that is, to the Common Set of Natural SQL syntax) or impose restrictions when used with DB2 (see also SQL Statements in the Natural Statements documentation).

atom

An atom can be either a parameter (that is, a Natural program variable or host variable) or a constant. When running dynamically, however, the use of host variables is restricted by DB2. For further details, refer to the relevant literature on DB2 by IBM.

comparison

The following three comparison operators are specific to DB2 and belong to the Natural Extended Set.

¬ =
¬ >
¬ <

factor

The following three factors are specific to DB2 and belong to the Natural Extended Set:

special-register
scalar-function (*scalar-expression, ...*)
scalar-expression unit
case-expression

scalar-function (???)

A scalar function (ohne Bindestrich, lt. IBM???) is a built-in function that can be used in the construction of scalar computational expressions. Scalar functions are specific to DB2 and belong to the Natural Extended Set.

The scalar functions supported are listed in alphabetical order:

A - H	I - R	S - Z
ABS	IDENTITY_VAL_LOCAL	SECOND
ABSVAL	IFNULL	SIGN
ACOS	INSERT	SIN
ADD_MONTHS	INTEGER	SINH
ASIN	JULIAN_DAY	SMALLINT
ATAN	LAST_DAY	SPACE
ATAN2	LCASE	SQRT
ATANH	LEFT	STRIP
BLOB	LENGTH	SUBSTR
CCSID_ENCODING	LN	TAN
CEIL	LOCATE	TANH
CEILING	LOG	TIME
CHAR	LOG10	TIMESTAMP
CLOB	LOWER	TIMESTAMP_FORMAT
COALESCE	LTRIM	TO_CHAR
CONCAT	MAX	TO_DATE
COS	MICROSECOND	TRANSLATE
COSH	MIDNIGHT_SECONDS	TRUNC
DATE	MIN	TRUNC_TIMESTAMP
DAY	MINUTE	TRUNCATE
DAYOFMONTH	MOD	UCASE
DAYOFWEEK	MONTH	UPPER
DAYOFWEEK_ISO	MULTIPLY_ALT	VALUE
DAYOFYEAR	NEXT_DAY	VARCHAR
DAYS	NULLIF	VARCHAR_FORMAT
DBCLOB	POSSTR	VARGRAPHIC
DEC	POWER	WEEK
DECIMAL	QUARTER	WEEK_ISO
DEGREES	RADIANS	YEAR
DIGITS	RAISE_ERROR	
DOUBLE	RAND	
DOUBLE-PRECISION (Bindestrich???)	REAL	
	REPEAT	
EXP	REPLACE	
FLOAT	RIGHT	
FLOOR	ROUND	
GRAPHIC	ROUND_TIMESTAMP	
HEX	ROWID	
HOUR	RTRIM	

Each scalar function is followed by one or more scalar expressions in parentheses. The number of scalar expressions depends upon the scalar function. Multiple scalar expressions must be separated from one another by commas.

Example:

```
SELECT NAME
  INTO NAME
  FROM SQL-PERSONNEL
  WHERE SUBSTR ( NAME, 1, 3 ) = 'Fri'
       ...
```

column function (klein???mit Bindestrich erforderlich???)

The following column functions do not conform to standard SQL. They are specific to DB2 and belong to the Natural Extended Set. Column functions operate on a set of values that derive from an expression (???) and return the defined value (???) or the NULL value.

AVG
COUNT
COUNT_BIG
MAX
MIN
STDDEV
STDDEV_POP
STDDEV_SAMP
SUM
VAR
VAR_POP
VAR_SAMP
VARIANCE
VARIANCE_SAMP

scalar-operator

The concatenation operator (CONCAT or "||") does not conform to standard SQL. It is specific to DB2 and belongs to the Natural Extended Set.

special-register

The following special registers do not conform to standard SQL. They are specific to DB2 and belong to the Natural Extended Set:

CURRENT APPLICATION ENCODING SCHEME
CURRENT DATE
CURRENT_DATE (???)
CURRENT DEGREE
CURRENT FUNCTION PATH
CURRENT_LC_CTYPE (???)
CURRENT LC_CTYPE
CURRENT LOCALE LC_CTYPE
CURRENT OPTIMIZATION HINT
CURRENT PACKAGESET
CURRENT_PATH
CURRENT PRECISION
CURRENT RULES
CURRENT SQLID
CURRENT SERVER
CURRENT TIME
CURRENT_TIME (???)

CURRENT TIMESTAMP
CURRENT TIMEZONE
CURRENT_TIMEZONE (???)
USER

A reference to a special register returns a scalar value.

Using the command SET CURRENT SQLID, the creator name of a table can be substituted by the current SQLID. This enables you to access identical tables with the same table name but with different creator names.

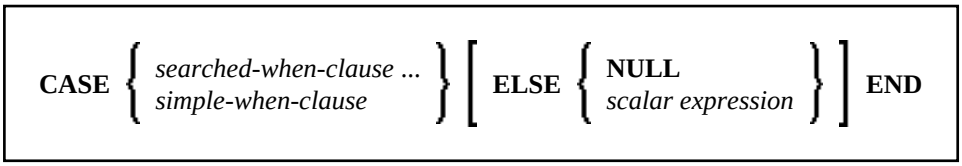
units

Units, also called durations, are specific to DB2 and belong to the Natural Extended Set.

The following units are supported:

DAY
DAYS
HOUR
HOURS
MICROSECOND
MICROSECONDS
MINUTE
MINUTES
MONTH
MONTHS
SECOND
SECONDS
YEAR
YEARS

case-expression



Case-expressions do not conform to standard SQL and are therefore supported by the Natural SQL Extended Set only.

Example:

```
DEFINE DATA LOCAL
01 #EMP
02 #EMPNO (A10)
02 #FIRSTNME (A15)
02 #MIDINIT  (A5)
02 #LASTNAME (A15)
02 #EDLEVEL   (A13)
02 #INCOME (P7)
END-DEFINE
SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME,
       (CASE WHEN EDLEVEL < 15 THEN 'SECONDARY'
             WHEN EDLEVEL < 19 THEN 'COLLEGE'
             ELSE             'POST GRADUATE'
        END ) AS EDUCATION, SALARY + COMM AS INCOME
```



```
        INTO
        #EMPNO, #FIRSTNME, #MIDINIT, #LASTNAME,
        #EDLEVEL, #INCOME
          FROM DSN8510-EMP
          WHERE (CASE WHEN SALARY = 0 THEN NULL
                                      ELSE SALARY / COMM
                                      END ) > 0.25
DISPLAY #EMP
END-SELECT
END
```

CALLDBPROC

Related documentation:
CALLDBPROC in SQL Statements in the Natural Statements documentation. ???

The CALLDBPROC statement allows you to call DB2 stored procedures. It supports the result set mechanism of DB2 Version 5 (???) and it enables you to call DB2 stored procedures written in Natural.

For a detailed description of the syntax of the statement, see the Natural Statements documentation.

If the CALLDBPROC statement is executed dynamically, all parameters and constants are mapped to the variables of the following DB2 SQL statement:

```
CALL :hv USING  DESCRIPTOR :sqlda  statement
```

:hv denotes a host variable containing the name of the procedure to be called and *:sqlda* is a dynamically generated sqlda describing the parameters to be passed to the stored procedure.

If the CALLDBPROC statement is executed statically, the constants of the CALLDBPROC statement are also generated as constants in the generated assembler SQL source for the DB2 precompiler.

If the SQLCODE created by the CALL statement indicates that there are result sets (SQLCODE +466 and +464), Natural for DB2 runtime executes a

```
DESCRIBE PROCEDURE :hv INTO :sqlda
```

statement in order to retrieve the result set locator values of the result sets created by the invoked stored procedure. These values are put into the RESULT SETS variables specified in the CALLDBPROC statement. Each RESULT SETS variable specified in a CALLDBPROC for which no result set locator value is present is reset to zero. The result set locator values can be used to read the result sets by means of the READ RESULT SET statement as long as the database transaction which created the result set has not yet issued a COMMIT or ROLLBACK.

If the result set was created by a cursor WITH HOLD, the result set locator value remains valid after a COMMIT operation.

Unlike other Natural SQL statements, CALLDBPROC enables you (optionally!) to specify a SQLCODE variable following the GIVING keyword which will contain the SQLCODE of the underlying CALL statement. If GIVING is specified, it is up to the Natural program to react on the SQLCODE (error message NAT3700 is not issued by the runtime).

Parameter data types supported by the CALLDBPROC statement:

Natural Format/Length	DB2 Data Type
A*n*	CHAR(*n*)
B2	SMALLINT
B4	INT
B*n* (*n* = not equal 2 or 4) (???)	CHAR(*n*)
F4	REAL
F8	DOUBLE PRECISION
I2	SMALLINT
I4	INT
N*nn.m*	NUMERIC(*nn+m,m*)
P*nn.m*	NUMERIC(*nn+m,n*)
G*n*	GRAPHIC(*n*)
A*n*/1:m	VARCHAR(*n*m*)
D	DATE
T	TIME (see also TIME below)

TIME ???

The format of the Natural parameter (???) **T** has a wider range (???) than the DB2 TIME data type (fomat???).

As a result, converting the **T** value into the TIME value, the date (???) fraction and the tenths of a second part of the relevant **T** field appear truncated in the equivalent (???) TIME field. Converting TIME into **T**, the date fraction (???) is reset to 0000-01-02 (???) and the tenths of a second part is reset to 0 in Natural.

CALLMODE=NATURAL

This parameter allows DB2 stored procedures written in Natural to be invoked Stored procedures written in Natural are Natural subprograms which execute in the stored procedure address space.

If the CALLMODE=NATURAL parameter is specified, an additional parameter describing the parameters passed to the Natural stored procedure is passed from the client, i.e. caller, to the server, i.e. stored procedure. The parameter is of format VARCHAR from the viewpoint of DB2. Therefore, every stored procedure written in Natural has to be defined in the SYSIBM.SYSPROCEDURES table (only applies to DB2 for OS/390 Version 5 and below) or with the CREATE PROCEDURE statement (DB2 UDB for OS/390 Version 6 and above) by using this VARCHAR parameter as the first in its PARMLIST row.

From the viewpoint of the caller, i.e. the Natural program, and from the viewpoint of the stored procedure, i.e. Natural subprogram, this additional parameter is invisible. It is passed as first parameter by the Natural for DB2 runtime and it is used as on the server side to build the copy of the passed data in the Natural thread and the corresponding CALLNAT statement. Additionally, this parameter serves as a container for error information created during execution of the Natural stored procedure by the Natural runtime. It also contains information on the library where you are logged on and the Natural subprogram to be invoked.

The following table describes the first parameter passed between the caller and the stored procedure if CALLMODE = NATURAL is specified.

NAME	FORMAT	PROCESSING MODE SERVER
STCBL	I2	Input (size of following information)
Procedure Information		
STCBLENG	A4	Input (printable STCBL)
STCBID	A4	Input (’STCB’)
STCBVERS	A4	Input (version of STCB ’310 ’)
STCBUSER	A8	Input (user ID)
STCBLIB	A8	Input (library)
STCBPROG	A8	Input (calling program)
STCBPSW	A8	Unused (password)
STCBSTNR	A4	Input (CALLDBPROC statement number)
STCBTCP	A8	Input (procedure called)
STCBPANR	A4	Input (number of parameters)
Error Information		
STCBERNR	A5	Output (Natural error number)
STCBSTAT	A1	Unused (Natural error status)
STCBLIB	A8	Unused (Natural error library)
STCBPRG	A8	Unused (Natural error program)
STCBLVL	A1	Unused (Natural error level)
STCBOTP	A1	Unused (error object type)
STCBEDYL	A2	Output (error text length)
STCBEDYT	A88	Output (error text)
	A100	Reserved for future use
Parameter Information		
FORMAT_DESCRIPTION	A variable	Input

The FORMAT_DESCRIPTION contains a description for each parameter passed to the stored procedure consisting of parameter type, format specification and length. Parameter type is the AD attribute of the CALLNAT statement as described in the Natural Statements documentation.

Each parameter has the following format description element in the FORMAT_DESC string

atl,*p*[,*d1*]....

where

- *a* is an attribute mark which specifies the parameter type:

Mark	Type	Equivalent AD Attribute	Equivalent DB2 Clause
M	modifiable	AD=M	INOUT
O	non-modifiable	AD=O	IN
A	input only	AD=A	OUT

- *t* is one of the following Natural format tokens:

t	Description	*l*	*p*	*dl*	Example
A	Alphanumeric	1-253	0	1-32767 or -	A30,0 or A30,0,10
N	Numeric unpacked	1-29	0-7	-	N10,3
P	Packed numeric	1-29	0-7	-	P13,4
I	Integer	2 or 4	0	-	I2,0
F	Floating point		0	-	I4,0
B	Binary		0	-	B23,0
D	Date	6	0	-	D6
T	Time	12	0	-	T12
L	Logical (unsupported)				

- *l* is an integer denoting the length/scale of the field. For numeric and packed numeric fields, *l* denotes the total number of digits of the field that is, the sum of the digits left and right of the decimal point. The Natural format N7.3 is, for example, represented by N10.3. See also the table above.
- *p* is an integer denoting the precision of the field. It is usually 0, except for numeric and packed fields where it denotes the number of digits right of the decimal point. See also the table above.
- *d1* is also an integer denoting the occurrences of the alphanumeric array (alphanumeric only). See also the table above.

This descriptive/control parameter is invisible to the calling Natural program and to the called Natural stored procedure, but it has to be defined in the parameter definition of the stored procedure row in the SYSIBM.SYSPROCEDURES table (only applies to DB2 for OS/390 Version 5 and below) or with the CREATE PROCEDURE statement (DB2 UDB for OS/390 Version 6 and above) and parameter style GENERAL or GENERAL WITH NULL. (???)

The following table shows the number of parameters which have to be defined in the SYSIBM.SYSPROCEDURES table (only applies to DB2 for OS/390 Version 5 and below) or with the CREATE PROCEDURE statement (DB2 UDB for OS/390 Version 6 and above) depending on the number of user parameters and whether the client (i.e. the caller of a stored procedure for DB2 for OS/390) and the server (i.e. the stored procedure for DB2 for OS/390) is written in Natural or in a different host language. *n* (???) denotes the number of 'user' (???) parameters.

Client\Server	Natural	not Natural
Natural	*n* + 1	*n* (CALLMODE=NONE) *n* + 1 (CALLMODE=NATURAL)
non-Natural	*n* + 1	N

Example issuing CALLDBPROC and READ RESULT SET statements:

```
DEFINE DATA LOCAL
1 ALPHA       (A8)
1 NUMERIC     (N7.3)
1 PACKED      (P9.4)
1 VCHAR       (A20/1:5) INIT     <'DB25SGCP'>
1 INTEGER2    (I2)
1 INTEGER4    (I4)
1 BINARY2     (B2)
1 BINARY4     (B4)
1 BINARY12    (B12)
1 FLOAT4      (F4)
1 FLOAT8      (F8)
1 INDEX-ARRAY (I2/1:11)
1 INDEX-ARRAY1(I2)
1 INDEX-ARRAY2(I2)
1 INDEX-ARRAY3(I2)
1 INDEX-ARRAY4(I2)
1 INDEX-ARRAY5(I2)
1 INDEX-ARRAY6(I2)
1 INDEX-ARRAY7(I2)
1 INDEX-ARRAY8(I2)
1 INDEX-ARRAY9(I2)
1 INDEX-ARRAY10(I2)
1 INDEX-ARRAY11(I2)
1 #RESP        (I4)
1 #RS1         (I4) INIT <99>
1 #RS2         (I4) INIT <99>
LOCAL
1 V1 VIEW OF SYSIBM-SYSTABLES
2 NAME
1 V2 VIEW OF SYSIBM-SYSPROCEDURES
2 PROCEDURE
2 RESULT_SETS
1 V (I2) INIT <99>
END-DEFINE
CALLDBPROC 'DAEFDB25.SYSPROC.SNGSTPC'  DSN8510-EMP
 ALPHA  INDICATOR :INDEX-ARRAY1
 NUMERIC INDICATOR :INDEX-ARRAY2
 PACKED  INDICATOR :INDEX-ARRAY3
 VCHAR(*) INDICATOR :INDEX-ARRAY4
 INTEGER2 INDICATOR :INDEX-ARRAY5
 INTEGER4 INDICATOR :INDEX-ARRAY6
 BINARY2  INDICATOR :INDEX-ARRAY7
 BINARY4  INDICATOR :INDEX-ARRAY8
 BINARY12 INDICATOR :INDEX-ARRAY9
 FLOAT4   INDICATOR :INDEX-ARRAY10
 FLOAT8   INDICATOR :INDEX-ARRAY11
  RESULT SETS #RS1 #RS2
 CALLMODE=NATURAL
READ (10) RESULT SET #RS2 INTO VIEW V2 FROM SYSIBM-SYSTABLES
WRITE 'PROC F RS  :' PROCEDURE 50T RESULT_SETS
END-RESULT
END
```

COMMIT

The SQL COMMIT statement indicates the end of a logical transaction and releases all DB2 data locked during the transaction. All data modifications are made permanent.

COMMIT is a synonym for the Natural END TRANSACTION statement.

No transaction data can be provided with the COMMIT statement.

If this command is executed from a stored procedure, Natural for DB2 does not execute the underlying commit operation. This allows the stored procedure to commit updates against non DB2 databases.

Under CICS, the COMMIT statement is translated into an EXEC CICS SYNCPOINT command. If the file server is used, an implicit end-of-transaction is issued after each terminal I/O. This is due to CICS-specific transaction processing in pseudo-conversational mode.

Under IMS/TM, the COMMIT statement is not translated into an IMS Checkpoint command, but is ignored. An implicit end-of-transaction is issued after each terminal I/O. This is due to IMS/TM-specific transaction processing.

Unless when used in combination with the WITH HOLD clause, a COMMIT statement must not be placed within a database loop, since all cursors are closed when a logical unit of work ends. Instead, it has to be placed outside such a loop or after the outermost loop of nested loops.

If an external program written in another standard programming language is called from a Natural program, this external program should not contain its own COMMIT command if the Natural program issues database calls, too. The calling Natural program should issue the COMMIT statement on behalf of the external program.

DELETE

Related documentation: DELETE in SQL Statements

Both the cursor-oriented or Positioned DELETE, and the non-cursor or Searched DELETE SQL statements are supported as part of Natural SQL; the functionality of the Positioned DELETE statement corresponds to that of the Natural DML DELETE statement.

With DB2, a table name in the FROM clause of a Searched DELETE statement can be assigned a *correlation-name* (kursiv ???). This does not correspond to the standard SQL syntax definition and therefore belongs to the Natural Extended Set.

The Searched DELETE statement must be used, for example, to delete a row from a self-referencing table, since with self-referencing tables a Positioned DELETE is not allowed by DB2.

INSERT

The INSERT statement is used to add one or more new rows to a table.

Since the INSERT statement can contain a select expression, all the DB2-specific syntactical items described above apply.

PROCESS SQL

The PROCESS SQL statement is used to issue SQL statements to the underlying database. The statements are specified in a *statement-string*, which can also include constants and parameters.

The set of statements which can be issued is also referred to as Flexible SQL and comprises those statements which can be issued with the SQL statement "EXECUTE".

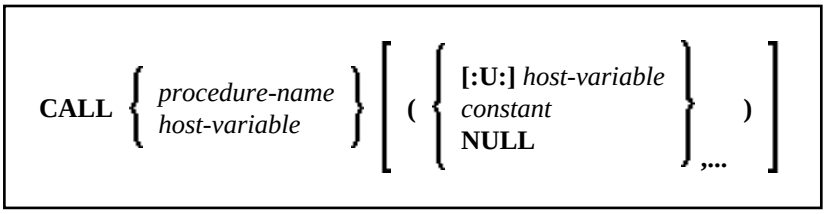
In addition, Flexible SQL includes the following DB2-specific statements:

CALL
CONNECT
SET APPLICATION ENCODING SCHEME
SET CONNECTION
SET CURRENT DEGREE
SET CURRENT LC_CTYPE
SET CURRENT OPTIMIZATION HINT
SET CURRENT PACKAGESET
SET CURRENT PATH
SET CURRENT PRECISION
SET CURRENT RULES
SET CURRENT SQLID
SET *host-variable= special-register* RELEASE

Note:
To avoid transaction synchronization problems between the Natural environment and DB2, the COMMIT and ROLLBACK statements must not be used within PROCESS SQL.

CALL

Natural for DB2 now supports the DB2 Version 4 (???) CALL statement by means of the PROCESS SQL statement. However, the syntax of the CALL statement is restricted as shown below.



The using descriptor parameter list format of the CALL statement is not supported.

Every host variable specified in the CALL parameter list should be prefixed with :U or the prefix should be omitted, regardless how the parameters are defined in the stored procedures parameter list. To use :G as *host-variable* prefix is strictly forbidden.

Example:

```
PROCESS SQL DB2-DDM

  <<CALL DB2PROC
         (:U:#USER,
          :U:#DATE,
          'ALPHA',
          NULL
         )
  >>
```

DB2PROC is a procedure name to be defined as a stored procedure in DB2.

#USER, #DATE are Natural variables.

'ALPHA' is a literal.

NULL is a keyword representing the NULL value.

Whether data are returned by the stored procedure called is only determined by the definition of the call parameter as defined for the stored procedure in DB2.

READ RESULT SET

The READ RESULT SET statement reads a result set created by a stored procedure that was invoked by a CALLDBPROC statement. For syntactical details, see the relevant sections in the Natural Statements documentation.

For details on how to specify the scroll direction for *scroll-hv*, see the SELECT statement in the Natural Statements documentation.(nix zu finden unter SM/Natural for SQL statements/SELECT???)

ROLLBACK

The SQL ROLLBACK statement undoes all database modifications made since the beginning of the last logical transaction. Logical transactions can start either after the beginning of a session or after the last COMMIT/END TRANSACTION or ROLLBACK/BACKOUT TRANSACTION statement. All records held during the transaction are released.

ROLLBACK is a synonym for the Natural statement BACKOUT TRANSACTION.

If this command is executed from a stored procedure in a user written Natural program, Natural for DB2 executes the underlying rollback operation. This sets the caller into a must-rollback state. If this command is executed from a stored procedure on behalf of the Natural error processing (implicit ROLLBACK), Natural for DB2 does not execute the underlying rollback operation, thus allowing the caller to receive the original Natural error.

Under CICS, the ROLLBACK statement is translated into an EXEC CICS ROLLBACK command. However, if the file server is used, only changes made to the database since the last terminal I/O are undone. This is due to CICS-specific transaction processing in pseudo-conversational mode.

Under IMS/TM, the ROLLBACK statement is translated into an IMS Rollback (ROLB) command. However, only changes made to the database since the last terminal I/O are undone. This is due to IMS/TM-specific transaction processing.

As all cursors are closed when a logical unit of work ends, a ROLLBACK statement must not be placed within a database loop; instead, it has to be placed outside such a loop or after the outermost loop of nested loops.

If an external program written in another standard programming language is called from a Natural program, this external program should not contain its own ROLLBACK command if the Natural program issues database calls, too. The calling Natural program should issue the ROLLBACK statement on behalf of the external program.

SELECT

Below is information on

- Cursor-Oriented Selection
- OPTIMIZE FOR integer ROWS Clause
- WITH Clauses

Cursor-Oriented Selection

Like the Natural FIND statement, the cursor-oriented SELECT statement is used to select a set of rows (records) from one or more DB2 tables, based on a search criterion. Since a database loop is initiated, the loop must be closed by a LOOP (reporting mode) or END-SELECT statement. With this construction, Natural uses the same loop processing as with the FIND statement.

In addition, no cursor management is required from the application program; it is automatically handled by Natural.

OPTIMIZE FOR integer ROWS Clause

The OPTIMIZE FOR *integer* ROWS clause is used to inform DB2 in advance of the number (*integer*) of rows to be retrieved from the result table. Without this clause, DB2 assumes that all rows of the result table are to be retrieved and optimizes accordingly.

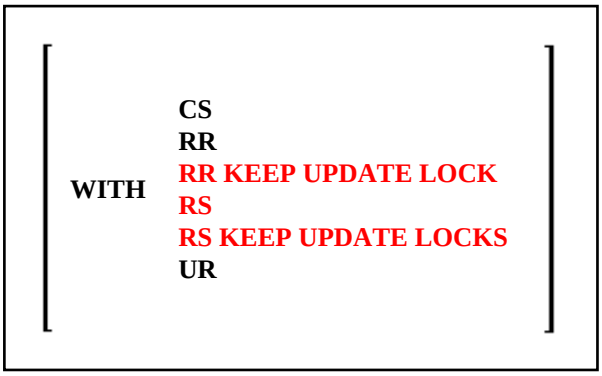
This optional clause is useful if you know how many rows are likely to be selected, because optimizing for *integer* rows can improve performance if the number of actually selected rows does not exceed the *integer* value (which can be in the range from 0 to 2147483647).

Example:

```
SELECT name INTO #name FROM table
WHERE AGE = 2 OPTIMIZE FOR 100 ROWS
```

WITH Clauses

WITH - Isolation Level



This WITH clause allows you to specify an explicit isolation level with which the statement is to be executed. The following options are provided:

Option	Meaning
CS	Cursor stability
RR	Repeatable Read
RS	Read Stability
RS KEEP UPDATE LOCKS	Only valid if specified in a FOR UPDATE OF statement.(???) Read Stability and retaining update locks.
RR KEEP UPDATE LOCKS	Only valid if specified in a FOR UPDATE OF statement.(???) Repeatable Read and retaining update locks.
UR	Uncommitted Read

WITH UR can only be specified within a SELECT statement and when the table is read-only. The default isolation level is determined by the isolation of the package or plan into which the statement is bound. The default isolation level also depends on whether the result table is read-only or not. To find out the default isolation level, refer to the IBM literature.

Note:
This option also works for non-cursor selection.

QUERYNO (Unterpunkt zu SELECT

[QUERYNO *integer*]

The QUERNO clause specifies the number of queries to be processed and traced for recodings as specified withthe EXPLAIN statement ??? The number of queries is used as QUERYNO in the plan table for the rows that contain information about this statement.

FETCH FIRST *integer* **ROWS ONLY**

[FETCH FIRST{ *integer* } { ROWS} ONLY]

[1] ROW???

The FETCH FIRST clause limits the number of rows to be fetched. It improves the performance of queries with potentially large result sets if only a limited number of rows is needed.

WITH HOLD

[WITH HOLD]

The WITH HOLD clause is used to prevent cursors from being closed by a commit operation within database loops. If WITH HOLD is specified, a commit operation commits all the modifications of the current logical unit of work, but releases only locks that are not required to maintain the cursor. This optional clause is mainly useful in batch mode; it is ignored in CICS pseudo-conversational mode and in IMS message-driven programs.

Example:

```
SELECT name INTO #name FROM table
WHERE AGE = 2 WITH HOLD
```

WITH RETURN

[WITH RETURN]

The WITH RETURN clause is used to create result sets. Therefore it should only be used in programs which operate as stored procedure. If the WITH RETURN clause is specified in a SELECT statement, the underlying cursor remains open when the associated processing loop is left, except when the processing loop had read all rows of the result set itself. During first execution of the processing loop, only the cursor is opened. The first row is not yet fetched. This allows the Natural program to return a full result set to the caller of the stored procedure. It is up to the Natural program to decide how many rows are processed by the program itself and how many unprocessed rows are returned to the caller of the stored procedure. If it wants to process rows of the select operation itself, it should code

```
IF *counter =1 ESCAPE TOP END-IF
```

in order to avoid processing of the first "empty row" in the processing loop. If it decides by some criteria of its own to terminate its own processing of rows, it should code

```
If condition ESCAPE BOTTOM END-IF
```

If the program reads all rows of the result set, the cursor is closed and no result set is returned for this SELECT WITH RETURN to the caller of the stored procedure.

The following programs are examples for retrieving full result sets (Example 1) and partial result sets (Example 2).

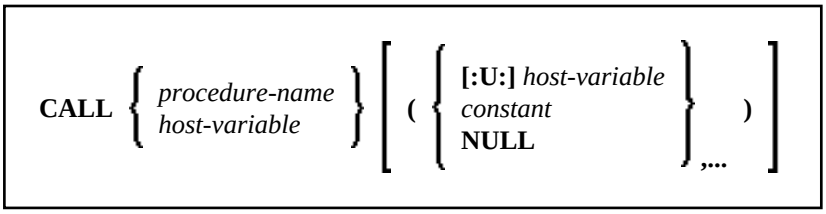
Example 1:

```
DEFINE DATA LOCAL
. . .
END DEFINE
*
*  Return all rows of the result set
*
SELECT * INTO VIEW V2
                    FROM SYSIBM-SYSROUTINES
                    WHERE RESULT_SETS > 0
                    WITH RETURN
ESCAPE BOTTOM
END-SELECT
END
```

Example 2:

```
DEFINE DATA LOCAL
. . .
END DEFINE
*
*  Read the first two rows and return the rest as result set
*
SELECT * INTO VIEW V2
                    FROM SYSIBM-SYSROUTINES
                    WHERE RESULT_SETS > 0
                    WITH   RETURN
WRITE PROCEDURE *COUNTER
IF *COUNTER = 1 ESCAPE TOP END-IF
IF *COUNTER =3 ESCAPE BOTTOM END-IF
END-SELECT
END
```

WITH INSENSITIVE/SENSITIVE (????)



[WITH {INSENSITIVE SCROLL} [.] *scroll_hv* **[GIVING [.]** *sqlcode***]]**

[WITH {SENSITIVE STATIC SCROLL} [.] *scroll_hv* **[GIVING [.]** *sqlcode***]]**

The WITH SENSITIVE/SENSITIVE clause is used to process (???) DB2 scrollable cursors. With scrollable cursors, an NDB (???) application can position on any row in a result set. For non-scrollable cursors, the data can only be read sequentially, from top to bottom. See also....????

NDB moves (???) the result sets of INSENSITIVE and SENSITIVE STATIC scrollable cursors into a temporary database space when the relevant cursos is opened. After scrollable cursors have been openend, the NDB (???) application can access and read (???) any row of the result set at any time in any direction (???).

Non-Cursor Selection - SELECT SINGLE

The Natural statement SELECT SINGLE provides the functionality of a non-cursor selection (singleton SELECT); that is, a select expression that retrieves at most one row without using a cursor.

Since DB2 supports the singleton SELECT command in static SQL only, in dynamic mode, the Natural SELECT SINGLE statement is executed like a set-level SELECT statement, which results in a cursor operation. However, Natural checks the number of rows returned by DB2. If more than one row is selected, a corresponding error message is returned.

UPDATE

Both the cursor-oriented or Positioned UPDATE, and the non-cursor or Searched UPDATE SQL statements are supported as part of Natural SQL. Both of them reference either a table or a Natural view.

With DB2, the name of a table or Natural view to be referenced by a Searched UPDATE can be assigned a *correlation-name*. This does not correspond to the standard SQL syntax definition and therefore belongs to the Natural Extended Set.

The Searched UPDATE statement must be used, for example, to update a primary key field, since DB2 does not allow updating of columns of a primary key by using a Positioned UPDATE statement.

Note:
If you use the SET * notation, all fields of the referenced Natural view are added to the FOR UPDATE OF and SET lists. Therefore, ensure that your view contains only fields which can be updated; otherwise, a negative *sqlcode* (oder SQLCODE ???) is returned by DB2.

